Long fMinStroke, fMaxStroke

These two fields are hints to the controller to help it efficiently determine the value of nStrokes for a unit that is a parent of this unit in the future. Note that nStrokes does not necessary equal maxStroke-minStroke.

Methods

static TUnit *Make(TDomain *domain, ULong type, TArray *areaList);

Make creates a new TUnit. The domain parameter indicates the domain that actually created the unit. If the unit was not created by a domain, this field may be set to 0. The type parameter is placed in the type field of the unit, and the areaList parameter is placed in the fareaList field of the unit. This is not a virtual method, so subclasses of TUnit are free to redefine the interface to their Make methods.

void IUnit(TDomain *domain, ULong type, TArray *areaList);

IUnit is called by Make to initialize the unit. Its parameters are the same as those passed to Make. IUnit should be called by the initialize methods of subclasses of TUnit. This is not a virtual method, so subclasses of TUnit are free to redefine the interface to their initialize methods.

virtual void Dispose(void);

This method is called to dispose of the TUnit object. Subclasses will only need to override this method if they allocate data structure of their own that is stored directly in the unit. Once an object that is subclassed from TUnit has been put into the recognition system (using NewGroup or NewClassification), you should never call this method directly. Instead, you should call ClaimUnits(unit), and the unit will be disposed of at a safe time.

virtual void IDispose(void);

virtual ULong SizeInBytes(void);

This method is for debugging purposes only. It should return the size, in bytes occupied by this unit, all of its owned structures, including its interpretations, but not including its subunits. A subclass of TUnit only needs to override this method if it stores allocated structures in the unit itself.

virtual void Clone(void);

This method causes the reference count in the unit to be incremented. This method should never need to be overwritten. Every call to Clone must be balanced by a call to Dispose, or the object will never be disposed.

virtual flag Release(void);

This method should be called by each Dispose method to determine whether or not to actually dispose of the object. It returns true if the object should be disposed, otherwise the Dispose method can simply return.

virtual ULong SubCount(void);

If the unit is a TUnit, SubCount always returns 0. If the unit is a subclass of TSIUnit, then SubCount returns the number of elements in the subs array.

virtual ULong InterpretationCount(void);

If the unit is a TUnit, this method returns 0. If the unit is a subclass of TSIUnit, then it returns the number of interpretations in the interpretations array. If your subclass of TUnit stores a single interpretation within the unit itself, then it should probably return 1 and implement all the appropriate interpretation methods.

virtual ULong GetBestInterpretation(void);

This method searches through the list of interpretations and returns the index of the one with the best score. You probably won't need to override this method.

virtual void Dump(TMsg *msg);

Dump is used for debugging purposes; see the TMsg class for a description of its use.

virtual void DumpName(TMsg *msg);

DumpName is used for debugging purposes; see the TMsg class for a description of its use.

virtual void ClaimUnit(TUnitList *);

ClaimUnit is called by the TController::ClaimUnits method; it shouldn't need to be called directly. If the unit is a TSIUnit, ClaimUnit calls itself recursively on all of its subunits, and then marks itself as claimed (setting the claimUnit flag in the flags field). If the unit is a TUnit, then it simply marks itself as claimed.

An application should claim a unit only if accepts an interpretation of the unit that is passed to it by the Arbiter. It should first extract all desired information from the unit or its interpretation, and then call ClaimUnits. Once ClaimUnits has been called on a unit, the unit is volatile and may disappear.

virtual void Invalidate(void);

Invalidate sets the invalid unit flag in the unit's flags field. A recognizer might call Invalidate after deciding that a particular grouping that it had previously proposed has become invalid. For instance if a single vertical stroke is categorized as a character unit with interpretation 'I', and a subsequent stroke crosses it, the recognizer might invalidate the 'I' unit and create a 't' unit instead.

The next set of methods are just for field access. They are not virtual functions so they can't be overwritten.

TDomain *GetDomain(void);

Call GetDomain to obtain a pointer to the domain that created the TSIUnit.

ULong GetDelay(void):
void SetDelay(ULong);
ULong GetType(void);
void SetType(ULong);
ULong GetPriority(void);
void SetPriority(ULong priority);
ULong GetTime(void)
void SetTime(ULong);
TArray *GetAreas(void);
void SetAreas(TArray *);
void SetBBox(rectangle *r);
rectangle *GetBBox(rectangle *r);
Long CheckOverlap(TUnit *a, TUnit *b);// return overlap status of two units
Long CountStrokes(TUnit *a);
Long CountOverlap(TUnit *a, TUnit *b);
void MarkStrokes(TUnit *a, char *ap, Long min);

While this invention has been described in terms of several preferred embodiments, it is contemplated that alterations, modifications and permutations thereof will become apparent to those skilled in the art upon a reading of the specification and study of the drawings. Furthermore, certain terminology has been used for the purposes of descriptive clarity, and not to limit of the present invention. It is therefore intended that the following appended claims include all such alterations, modifications and permutations as fall within the true spirit and scope of the present invention.

We claim:

1. A system for recognizing user input to a computer from a user input device, comprising:

input means for receiving user input data in the form of a